USER DEFINED FUNCTIONS AND STORED PROCEDURES

I. More examples on user defined functions

❖ Scalar function:

```
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
-- Returns the stock level for the product at location ID 6
BEGIN
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID
        AND p.LocationID = '6';
     IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
```

Some ways to execute scalar function:

```
SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS
CurrentSupply
FROM Production.Product
WHERE ProductModelID BETWEEN 75 and 80;

Select dbo.ufnGetInventoryStock(78);
```

Another example:

```
-- Example:
CREATE FUNCTION [dbo].[ufn_GetContactOrders](@BusinessEntityID int)
RETURNS varchar(500)
AS
BEGIN
    DECLARE @Orders varchar(500)

    SELECT @Orders = COALESCE(@Orders + ', ', '') + CAST(SalesOrderID as
varchar(10))
    FROM Sales.SalesOrderHeader
    WHERE BusinessEntityID = @BusinessEntityID

    RETURN (@Orders)
END
```

//Use the scalar function:

```
SELECT BusinessEntityID, dbo.ufn_GetContactOrders(BusinessEntityID) FROM
Person.Person
WHERE BusinessEntityID between 100 and 105 -- Output below

-- Used while defining a computed column while creating a table.
```

```
CREATE TABLE tempCustOrders (CustID int, Orders as
(dbo.ufn_GetContactOrders(CustID)))

INSERT INTO tempCustOrders (CustID)
SELECT BusinessEntityID FROM Person.Person
WHERE BusinessEntityID between 100 and 105

SELECT * FROM tempCustOrders -- Output below

DROP TABLE tempCustOrders
```

- ❖ **Table valued functions:**

- Inline table value function

```
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
(
    SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'
    FROM Production.Product AS P
    JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
    JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID
    JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
    WHERE C.StoreID = @storeid
    GROUP BY P.ProductID, P.Name
);
```

To execute inline table value function:

```
SELECT * FROM Sales.ufn_SalesByStore (602);
```

Other example:

```
CREATE FUNCTION [dbo].[ufn_itv_GetContactSales](@BusinessEntityID int)
RETURNS TABLE
AS
RETURN (
    SELECT h.[BusinessEntityID], h.[SalesOrderID], p.[ProductID], p.[Name],
h.[OrderDate], h.[DueDate],
    h.[ShipDate], h.[TotalDue], h.[Status], h.[SalesPersonID]
    FROM Sales.SalesOrderHeader AS h
    JOIN Sales.SalesOrderDetail AS d ON d.SalesOrderID = h.SalesOrderID
    JOIN Production.Product AS p ON p.ProductID = d.ProductID
    WHERE BusinessEntityID = @BusinessEntityID )

--// Usage:
SELECT * FROM ufn_itv_GetContactSales(100)
```

- ❖ **Multi statement table value functions**

```
CREATE FUNCTION dbo.ufn_FindReports (@InEmpID INTEGER)
```

```
RETURNS @retFindReports TABLE
(
    EmployeeID int primary key NOT NULL,
    FirstName nvarchar(255) NOT NULL,
    LastName nvarchar(255) NOT NULL,
    JobTitle nvarchar(50) NOT NULL,
    RecursionLevel int NOT NULL
)
--Returns a result set that lists all the employees who report to the
--specific employee directly or indirectly.*/
AS
BEGIN
WITH EMP_cte(EmployeeID, OrganizationNode, FirstName, LastName, JobTitle,
RecursionLevel) -- CTE name and columns
    AS (
        SELECT e.BusinessEntityID, e.OrganizationNode, p.FirstName,
p.LastName, e.JobTitle, 0 -- Get the initial list of Employees for Manager n
        FROM HumanResources.Employee e
INNER JOIN Person.Person p
ON p.BusinessEntityID = e.BusinessEntityID
        WHERE e.BusinessEntityID = @InEmpID
        UNION ALL
        SELECT e.BusinessEntityID, e.OrganizationNode, p.FirstName,
p.LastName, e.JobTitle, RecursionLevel + 1 -- Join recursive member to anchor
        FROM HumanResources.Employee e
            INNER JOIN EMP_cte
            ON e.OrganizationNode.GetAncestor(1) = EMP_cte.OrganizationNode
INNER JOIN Person.Person p
ON p.BusinessEntityID = e.BusinessEntityID
        )
-- copy the required columns to the result of the function
   INSERT @retFindReports
   SELECT EmployeeID, FirstName, LastName, JobTitle, RecursionLevel
   FROM EMP_cte
   RETURN
END;
GO
```

//To execute multi statement table value function

```
SELECT EmployeeID, FirstName, LastName, JobTitle, RecursionLevel
FROM dbo.ufn_FindReports(1);
```

Other example:

```
CREATE FUNCTION [dbo].[ufn_mtv_GetContactSales](@BusinessEntityID int)
RETURNS @retSalesInfo TABLE (
    [BusinessEntityID] INT NOT NULL,
    [SalesOrderID] INT NULL,
    [ProductID] INT NULL,
    [Name] NVARCHAR(50) NULL,
    [OrderDate] DATETIME NULL,
    [DueDate] DATETIME NULL,
```

```
    [ShipDate] DATETIME NULL,
    [TotalDue] MONEY NULL,
    [Status] TINYINT NULL,
    [SalesPersonID] INT NULL)
AS
BEGIN
    IF @BusinessEntityID IS NOT NULL
    BEGIN
        INSERT @retSalesInfo
        SELECT h.[BusinessEntityID], h.[SalesOrderID], p.[ProductID], p.[Name],
h.[OrderDate], h.[DueDate],
               h.[ShipDate], h.[TotalDue], h.[Status], h.[SalesPersonID]
        FROM Sales.SalesOrderHeader AS h
        JOIN Sales.SalesOrderDetail AS d ON d.SalesOrderID = h.SalesOrderID
        JOIN Production.Product AS p ON p.ProductID = d.ProductID
        WHERE BusinessEntityID = @BusinessEntityID
    END
    -- Return the recordsets
    RETURN
END

--// Usage:
SELECT * FROM ufn_mtv_GetContactSales(100)
```

II. Stored procedures

MS SQL Server has four types of stored procedures:

- User-defined
- System
- Temporary
- Extended user-defined

The extended user-defned stored procedures have been replaced with common language runtime (CLR) procedures If you want more information on CLR stored procedures, visit the “CLR Stored Procedure” on microsoft tutorial.

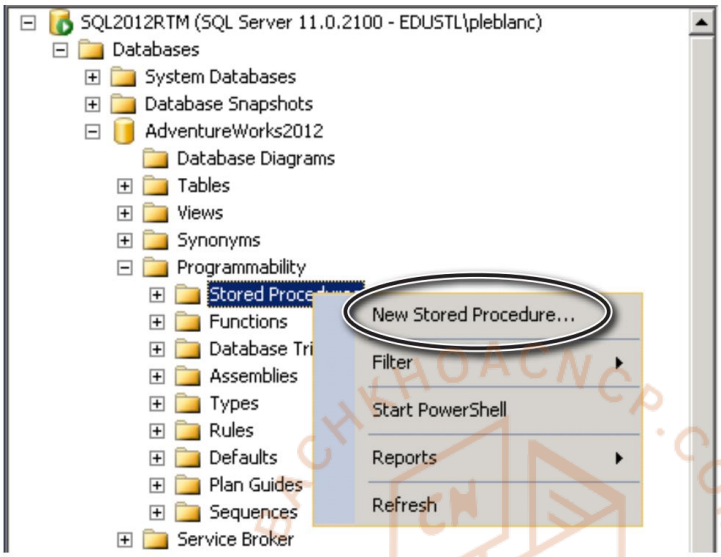
We focus on user-defined store procedures

```
-- Stored Procedure Syntax
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
 [ { @parameter [ type_schema_name. ] data_type }
 [ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
 ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
<procedure_option> ::=
 [ ENCRYPTION ]
 [ RECOMPILE ]
 [ EXECUTE AS Clause ]
```

When creating a stored procedure, similar to a function, you are able to set several options. However, only the *procedure_name* and the actual *sql_statement*(s) are required.

To create stored procedures:

- Expand the Programmability folder
- Right-click the Stored Procedures folder and select New Stored Procedure



- Open a new query window and paste in the following code:

```
USE AdventureWorks2012;
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author: Patrick LeBlanc
-- Create date: 6/9/2012
-- Description: <Description,,> Get PurchaseOrder Information
-- ============================================
CREATE PROCEDURE dbo.PurchaseOrderInformation
AS
BEGIN
     SELECT
          poh.PurchaseOrderID, pod.PurchaseOrderDetailID,
          poh.OrderDate, poh.TotalDue, pod.ReceivedQty, p.Name ProductName
     FROM Purchasing.PurchaseOrderHeader poh
     INNER JOIN Purchasing.PurchaseOrderDetail pod
          ON poh.PurchaseOrderID = pod.PurchaseOrderID
     INNER JOIN Production.Product p
          ON pod.ProductID = p.ProductID
END
GO
```

To execute a stored procedure using T-SQL, you use the EXECUTE keyword The syntax is as follows:

EXECUTE | EXEC procedure_name [parameter1, parameter2, n…]

You can change the names and data types of each column in the result set or redefne the result set. The syntax is as follows:

```
EXECUTE | EXEC procedure_name [parameter1, parameter2, n…]
WITH RESULT SETS
(
( [column_definition1, column_definition2, n…])
)
```

For example, try to see differences:

//execute

```
USE AdventureWorks2012
EXEC dbo.PurchaseOrderInformation
```

//execute with resultsets

```
USE AdventureWorks2012;
EXEC dbo.PurchaseOrderInformation
WITH RESULT SETS
(
(
[Purchase Order ID] int,
[Purchase Order Detail ID] int,
[Order Date] datetime,
[Total Due] Money,
[Received Quantity] float,
[Product Name] varchar(50)
)
)
```

Parameterized stored procedures

Similar to functions, stored procedures can include parameters as part of their code. Creating a stored procedure with parameters allows the calling programs to pass values into the procedure. Parameters in stored procedures differ from those in functions in that you can specify direction, whether it is an input or output parameter In other words, you can specify whether the parameter will accept a value (input) or whether it will return a value (output).

EXEC [dbo].[PurchaseOrderInformation] @parameter1 = 1, @parameter2 = default, @parameter3 = null

Example 1 - simple stored procedure
This first example creates a simple stored procedure that gets the TOP 1 record from the Person.Person table.
CREATE PROCEDURE uspGetContact AS SELECT TOP 1 BusinessEntityID, FirstName, LastName FROM Person.Person
After the above has been created use the command below to execute this stored procedure.
EXEC uspGetContact

Example 2 - stored procedure with a parameter
This next example is a modification of the first example, but this time adding a parameter that is passed into the procedure to dynamically select the records. Instead of using CREATE PROCEDURE we are using ALTER PROCEDURE to modify the procedure that we created in Example 1 instead of dropping it first and then recreating it.
ALTER PROCEDURE uspGetContact @LastName NVARCHAR(50) AS SELECT TOP 1 BusinessEntityID, FirstName, LastName FROM Person.Person WHERE LastName = @LastName
Below shows two different ways the stored procedure can be run. The first example just passes the parameter value we want to use and the second example also includes the parameter name along with the value. You can run the stored procedure with either one of these commands.
EXEC uspGetContact 'Alberts' EXEC uspGetContact @LastName='Alberts'

Example 3 - stored procedure with a parameter and output parameter
In this example we have both an input parameter as well as an OUTPUT parameter. The output parameter will be used to pass back the BusinessEntityID that we are looking up in the stored procedure. This output parameter will then be used to select the persons BusinessEntityID, FirstName and LastName along with any address records for this person. Again we are altering the stored procedure uspGetContact and then secondly we are running the next set of code that executes procedure uspGetContact and then based on the return value it gets

it will also query for the persons name and address info.

```
ALTER PROCEDURE uspGetContact @LastName NVARCHAR(50), @BusinessEntityID INT
 output
AS
SELECT TOP 1 @BusinessEntityID = c.BusinessEntityID
   FROM HumanResources.Employee a
INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b. BusinessEntityID
      INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
      INNER JOIN Person.Address d ON b.AddressID = d.AddressID
WHERE c.LastName = @LastName
```

After the stored procedure has been altered run the below block of code. This will execute the above stored procedure and if the BusinessEntityID has a value it will also return the person and address info.

```
DECLARE @BusinessEntityID INT
SET @BusinessEntityID = 0
EXEC uspGetContact @LastName='Smith', @BusinessEntityID=@BusinessEntityID OUTPUT
IF @BusinessEntityID <> 0
BEGIN
   SELECT BusinessEntityID, FirstName, LastName
   FROM Person.Person
   WHERE BusinessEntityID = @BusinessEntityID

   SELECT d.AddressLine1, d.City, d.PostalCode
   FROM HumanResources.Employee a
      INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b.
BusinessEntityID
       INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
      INNER JOIN Person.Address d ON b.AddressID = d.AddressID
   WHERE c.BusinessEntityID = @BusinessEntityID
END
```

Example 4 - stored procedure using the RAISERROR statement

In this example we are combining the two steps in Example 3 into one stored procedure. The first step is to get the BusinessEntityID and then the second part of the procedure will lookup the persons name and address info. We also added in code to use the RAISERROR statement to return an error if no records are found.

This is then being run twice to show what it looks like when data is found and when no data is found. The RAISERROR statement can be used to control how your application handles no data or any other error that may occur.

```
ALTER PROCEDURE uspGetContact @LastName NVARCHAR(50)
AS
```

```
DECLARE @BusinessEntityID INT
SELECT TOP 1 @BusinessEntityID = c.BusinessEntityID
FROM HumanResources.Employee a
    INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
      INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b.
BusinessEntityID
      INNER JOIN Person.Address d ON b.AddressID = d.AddressID
WHERE c.LastName = @LastName

IF @@ROWCOUNT > 0
BEGIN
  SELECT BusinessEntityID, FirstName, LastName
  FROM Person.Person
  WHERE BusinessEntityID = @BusinessEntityID

  SELECT d.AddressLine1, d.City, d.PostalCode
  FROM HumanResources.Employee a
      INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b.
BusinessEntityID
      INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
      INNER JOIN Person.Address d ON b.AddressID = d.AddressID
  WHERE c.BusinessEntityID = @BusinessEntityID
END
ELSE
BEGIN
  RAISERROR ('No record found',10,1)
END
```

```
EXEC uspGetContact @LastName='Walters'
EXEC uspGetContact @LastName='Job'
```

Example 5 - stored procedure with a separate calling stored procedure

Here is another example where we have two stored procedures. The first stored procedure uspFindContact lookups the first record that has an address record and then returns the BusinessEntityID to the calling stored procedure to again display the person and address info.

```
CREATE PROCEDURE uspFindContact @LastName NVARCHAR(50), @BusinessEntityID I
NT output
AS
SELECT TOP 1 @BusinessEntityID = c.BusinessEntityID
  FROM HumanResources.Employee a
          INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b.
BusinessEntityID
INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
      INNER JOIN Person.Address d ON b.AddressID = d.AddressID
```

```
WHERE c.LastName = @LastName
```

The code below does an lter of the uspGetContact stored procedure that calls uspFindContact and returns the recordsets.

```
ALTER PROCEDURE uspGetContact @LastName NVARCHAR(50)
AS
DECLARE @BusinessEntityID INT
SET @BusinessEntityID = 0

EXEC uspFindContact @LastName=@LastName, @BusinessEntityID=@BusinessEntityID OU
TPUT

IF @BusinessEntityID <> 0
BEGIN
  SELECT BusinessEntityID, FirstName, LastName
  FROM Person.Person
  WHERE BusinessEntityID = @BusinessEntityID

  SELECT d.AddressLine1, d.City, d.PostalCode
  FROM HumanResources.Employee a
        INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b.
BusinessEntityID
    INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
    INNER JOIN Person.Address d ON b.AddressID = d.AddressID
  WHERE c.BusinessEntityID = @BusinessEntityID
END
ELSE
BEGIN
  RAISERROR ('No record found',10,1)
END
```

```
EXEC uspGetContact @LastName='Walters'
EXEC uspGetContact @LastName='Job'
```

Example 6 - stored procedure with comments

This last example takes the uspGetContact stored procedure and adds comments to the code so you can see how comments work within a stored procedure. You can see that there are two ways that comments can be made. 1) using -- and 2) using /* to begin the comment block and */ to end the comment block. Other than that nothing else has changed.

```
ALTER PROCEDURE uspGetContact @LastName NVARCHAR(50)
AS
/* This is a sample stored procedure to show
  how comments work within a stored procedure */
```

```
-- declare variable
DECLARE @BusinessEntityID INT
-- set variable value
SET @BusinessEntityID = 0

-- execute stored proc and return BusinessEntityID value
EXEC uspFindContact @LastName=@LastName, @BusinessEntityID=@BusinessEntityID OU
TPUT

-- if BusinessEntityID does not equal 0 then return data else return error
IF @BusinessEntityID <> 0
BEGIN
  SELECT BusinessEntityID, FirstName, LastName
  FROM Person.Person
  WHERE BusinessEntityID = @BusinessEntityID

  SELECT d.AddressLine1, d.City, d.PostalCode
  FROM HumanResources.Employee a
          INNER JOIN Person.BusinessEntityAddress b ON a.BusinessEntityID = b.
BusinessEntityID
      INNER JOIN Person.Person c ON a.BusinessEntityID = c.BusinessEntityID
      INNER JOIN Person.Address d ON b.AddressID = d.AddressID
  WHERE c.BusinessEntityID = @BusinessEntityID
END
ELSE
BEGIN
  RAISERROR ('No record found',10,1)
END
```